Commodore

The Transactor

comments and bulletins concerning your COMMODORE PET™

Vol. 2

BULLETIN # 4

August 31, 1979

PET™ is a registered Trademark of Commodore Inc.

Commuted GOTO

Ever wanted to program a GOTO followed by an expression such as:

```
120 IF ST GOTO (ST * 10)
```

Normally PET does not allow this but Brad Templeton of Mississauga has submitted a machine language routine that will handle a computed GOTO. The program fits in only twelve bytes and can be placed in any part of memory where it won't get clobbered by BASIC. It accesses code in ROM and therefore has two versions, one for original ROM and another for upgrade ROM:

```
Original ROM:   JSR CE11     checks for comma
                             else SYNTAX ERROR
                JSR CCA4     evaluates expression
                JSR D6D0     integer? >=0 and <=63999
                JMP C7A0     jump to GOTO routine with result


Upgrade ROM:    JSR CDF8
                JSR CC8B     same as above
                JSR D6D2
                JMP C7B0
```

Because the program has no reference to itself, it can be placed anywhere, but for now we'll put it in the 2nd cassette buffer starting at 826 or hex 033A. Syntax for using the routine will be:

```
                SYS826,expression
or...           0%=826 : SYS0%, expression

e.g.            IF ST THEN SYS 0%, ST * 10
```

BASIC Loader

With the following modification, both of the above routines can be loaded into the 2nd cassette buffer and PET will decide which to use. This way, programs using the computed GOTO can be run with either ROMs.

```
                LDA #F202
                BMI #0D
N.ROMs:         JSR  CDF8
                JSR  CC8B
                JSR  D6D2
                JMP  C7B0
O.ROMs:         JSR  CE11
                JSR  CCA4
                JSR  D6D0
                JMP  C7A0
```

The following BASIC program will load the above:

```
100 FOR J = 826 TO 854
110 READ X
120 POKE J , X
130 NEXT
200 DATA 173 , 02 , 242 , 48 , 13
210 DATA 32 , 246 , 205 , 32 , 139 , 204 ,
    32 , 210 , 214 , 76 , 176 , 199
220 DATA 32 , 17 , 206 , 32 , 164 , 204 ,
    32 , 208 , 214 , 76 , 160 , 199
```

Test with the following:

```
10 G% = 826
20 ?"TEST"
30 SYS G%, 2 * 10
```

Many programmers like to indent their FOR-NEXT loops, to enhance readability. Up until now, this has only been possible by putting a colon (:) at the start of each line to be indented or spaced. For example:

```
10 FOR I=1 TO 10
20 : FOR J=1 TO 10
30 :   FOR K=1 TO 10
40 :
50 :     PRINT I,J,K
60 :
70 :   NEXT K
80 : NEXT J
90 NEXT I
```

This helps readability greatly, but you can go even further! By substituting SHIFTED(graphic) characters instead of colons, and using ▒ ▒ (graphic space graphic) to blank a line, the listing would be typed in like this (note: any shifted character can be substituted for the ▒):

```
10 FOR I=1 TO 10
20 ▒ FOR J=1 TO 10
30 ▒   FOR K=1 TO 10
40 ▒ ▒
50 ▒     PRINT I,J,K
60 ▒ ▒
70 ▒   NEXT K
80 ▒ NEXT J
90 NEXT I
```

This would list like this:

```
10 FOR I=1 TO 10
20   FOR J=1 TO 10
30     FOR K=1 TO 10
40
50       PRINT I,J,K
60
70     NEXT K
80   NEXT J
90 NEXT I
```

The same result is achieved, but the listing is cleaner. To use the screen editor, and retain this formatting, list the problem lines, put a ▒ after the line#, and edit as usual.

More PET "quirks" — Rick Ellis, Toronto, Canada.

Clear The Screen on your 8K PET and type in the following lines:

```
POKE 59468,14

1ØOp1,3:?"cs":X=63:Y=192

2ØFoI=ØTOX:I$=Ri(STr(I),1):?"chrv"TaI)Ch(I+Y)"cl";:
Ge#1,A$:?:?"chcdcdcd"TaX-I)A$:NeI

Li
```

(cs = Clear Screen)
(ch = Cursor Home)
(rv = Reverse)
(cl = Cursor Left)
(cd = Cursor Down)

(Type line #20 above as one continuous line).

Surprise ! Line 20 is over 100 characters long. Before you try to run the above program, check that your listed version reads as follows. If not, correct it now by moving the cursor up and correcting the version you typed in to match the above:

```
1Ø OPEN1,3:PRINT"cs":X=63:Y=192
2Ø FOR I=Ø TO X:
     I$=RIGHT$(STR$(I),1):
     PRINT"chrv" TAB(I) CHR$(I+Y) "cl";:
     GET#1,A$:
     PRINT:
     PRINT"chcdcdcd"TAB(X-I) A$:
     NEXT I
```

except of course you wont see it spaced out as above.

Now type:

```
Ru
```

The program now displays a character-string on screen lines 1 & 2, in REVERSE, and as it prints each character, reads it from the screen with the GET#1 command, and reprints it in reverse order below.

Try changing line #10 (yes, it's short enough!), so that Y = 64 and RUN again.

IF less Decisions

99% of all computer programs contain at least one decision making statement. The fundamental decision makers in PET BASIC are of course the IF-THEN and IF-GOTO statements. However, when a program performs a lot of tests or comparisons, it can become plagued with IF-THEN statements. Following are a few techniques for making decisions without 'IF'.

1. In real-time programs where GET is used to echo keyboard input onto the screen, some keys may need to be intercepted else cause undesirable effects, keys such as RVS, DEL, INST, CLR, etc. Also, other keys might want to be used as 'control' keys for initializing functions; keys such as RETURN, RVS, shifted RETURN, HOME, etc. Below is a routine which eliminates countless 'IFs'.

```
55000 C$ = "@##$%&*+/><"+CHR$(13):'CLR''HOME''RVS'
      'RVSOFF' " + CHR$('DEL') + CHR$('INST')
      + CHR$('RET') + CHR$('shRET')
55010 GET T$ : IF T$ = "" THEN 55010
55020 B = 0
55030 FOR J = 1 TO LEN (C$)
55040 A$ = MID$ (C$ , J , 1)
55050 IF A$ = T$ THEN B=J : J=LEN(C$)
55060 NEXT
55070 IF B = 0 THEN PRINT T$;:GOTO 55010
55080 ON B GOTO 60000,60100,60200,60300,
      60400,60500,60600,60700,60800,60900,
55090 ON B-10 GOTO 61000,61100,61200,61300
      61400,61500,61600,61700,61800,61900
```

This routine will PRINT any character not included in C$. A repeat-key could also be implemented with:

```
55070 IF B = 0 THEN PRINT T$;:POKE515,255
      :GOTO55010
```

2. See if you can determine what decisions the following two programs are making.

```
10 INPUT X , Y
20 PRINT ( X + Y - ABS( X - Y ))/2
30 GOTO 10


10 INPUT X , Y
20 PRINT ( X + Y + ABS( X - Y ))/2
30 GOTO 10
```

Modifications of the above routines (i.e. using a FOR-NEXT loop and array variables) might be useful in programs performing sorts.

3.
```
IF B = 0 THEN A = 32768
IF B = 1 THEN A = 1.259
IF B = 2 THEN A = 556.2
IF B = 3 THEN A = 400 * B
```

The above could continue forever depending on the possibilities for B. Try the following in direct mode on your PET:

```
    Type: B = 2
Now type: ? B = 0
```

PET will reply with 0 because B does not equal 0.

```
    Type: ? B  = 2
     and: ? B <> 0
```

In both cases PET will return a "-1" because the statements are true. This can be used most efficiently to replace the above IF-THEN statements:

```
A = -((B = 0)* 32768 + (B=1)* 1.259 + (B=2)* 556.2
    + (B=3) * 400 * B)
```

Since only one will be true, the others will be multiplied by zero and added. The negative sign in front changes the result back to positive.

4. This one uses the 'IF' statement but no comparison operator is used (i.e. >,=,<,<>). Try the following program.

```
10 INPUT X
20 IF X THEN ?"DID BRANCH":GOTO 10
30 ?"DID NOT BRANCH":GOTO 10
```

"DID BRANCH" occurs if X is anything but zero. On what condition does the following program branch:

```
10 INPUT X
20 IF NOT X AND 1 THEN ?"DID BRANCH":GOTO 10
30 ?"DID NOT BRANCH":GOTO 10
```

Disabling the STOP key.

It's useful to be able to disable the STOP key, so tha program cannot be accidentally (or deliberately) stopped

METHOD A is quick. Any cassette tape activity will rese the STOP key to its normal function, however.

METHOD A, Original ROM:

```
Disable the STOP key with POKE 537,136
Restore the STOP key with POKE 537,133
```

METHOD A, Upgrade ROM:

```
Disable the STOP key with POKE 144,49
Restore the STOP key with POKE 144,46
```

Method A also disconnects the co ter's clocks (TI and TI$). If you need these for timing in program, you should use method B.

METHOD B is slightly more lengthy, but does not disturb the clocks. This method prohibits cassette tape activity.

METHOD B, Original ROM:

```
100 R$="20>:??:9??8=09024<88>6"
110 FOR I=1 TO LEN(R$)/2
120 POKE I+900,ASC(MID$(R$,I*2-1))*16 +
          ASC(MID$(R$,I*2))-816 : NEXT I
```

After the above has run:

```
Disable the STOP key with POKE 538,3
Restore the STOP key with POKE 538,230
```

METHOD B, Upgrade ROM:

```
100 R$="20>:??:9??8=9;004<31>6"
110 FOR I=1 TO LEN(R$)/2
120 POKE I+844,ASC(MID$(R$,I*2-1))*16 +
          ASC(MID$(R$,I*2))-816 : NEXT I
```

After the above has run:

```
Disable the STOP key with POKE 144,77 : POKE 145,3
Restore the STOP key with POKE 145,230 : POKE 144,46
```

How they work: Both methods change the interrupt program which takes care of the keyboard, cursor, clocks and the stop key.

Method A simply skips the clock update and the stop key test.

Method B builds a small program into the second cassette buffer which performs the clock update and stop key test, but then nullifies the result of this test.

The little program in method B is contained in R$ in "pig hexadecimal" format. Machine language programmers would read this as: 20 EA FF (do clock update and stop key test) A9 FF 8D 9B 00 (cancel stop test result) 4C 31 E6 (continue with keyboard service, etc.)

Mortaald

A Fast Sort. Jim Butterfield, Toronto

When you need to sort a large array, sorting speed becomes important. Most simple sorts become very slow, since twice as many items will take four times as long to sort.

This fast sort is called "selective replacement"; it's classified as a tree type sort. It needs an index array, called I(J) here, which is twice the size of the items to be sorted. Memory can be saved, if needed, by making it an integer type array.

```
100 DIM I(200), N$(100), A$(100)
110 REM  SIMPLE INPUT ROUTINE - WRITE YOUR OWN FOR FILES
120 INPUT "HOW MANY ITEMS"; N
130   FOR J=0 TO N-1
140   INPUT "NAME";N$(J)
150   INPUT "ADDRESS";A$(J)
160   REM   INPUT OTHER DATA HERE IF DESIRED
170   NEXT J
200 REM   SORT STARTS HERE - INITIAL SCAN FINDS FIRST NUMBER
210 B=N-1 : FOR J=0 TO B : I(J)=J : NEXT J
220   FOR J=0 TO N*2-3 STEP 2
230   B=B+1 : I1=I(J) : I2=I(J+1)
240   GOSUB 700   :    REM  PERFORM COMPARISON
250   I(B)=I : NEXT J
300 REM   MAIN LOOP - OUTPUT NEXT VALUE
310 X=X-1 : C=I(B) : IF C<0 GOTO 800
320 REM  OUTPUT ITEM TO SCREEN, PRINTER, OR FILE AS DESIRED
330 PRINT N$(C),A$(C)
340 I(C)=X
350 REM  INNER LOOP TO FIND NEXT ITEM
360 C%=C/2 : J=C%*2 : C=N+C% : IF C>B GOTO 300
370 I1=I(J) : I2=I(J+1)
380 IF I1<0 THEN I=I2 : GOTO 410
390 IF I2<0 THEN I=I1 : GOTO 410
400 GOSUB 700 :  REM     PERFORM COMPARISON
410 I(C)=I : GOTO 350
700 REM     COMPARE TWO ITEMS - MODIFY TO FIT APPLICATION
710 I=I1 : IF N$(I2)<N$(I1) THEN I=I2
720 RETURN
800 STOP :   REM END OF SORT
```

As you get the sorted item at line 320, it's best to output it (or process it) on the spot. If some reason exists for completing the sort before going on to other processing, you'll find that index array I(J) contains information about the proper order for the data.

PET CRT Fix

James Yost of Somerville Massachusetts has written the Transactor with the following fix for all PET screens...

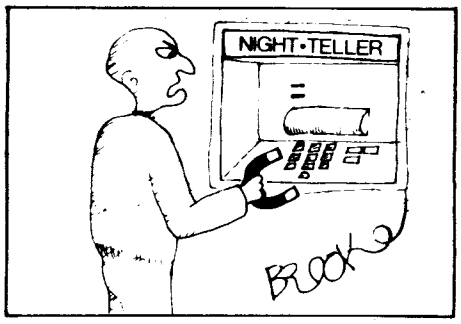
> ...I just discovered that the bright spot on power-down can be corrected very easily. The brightness pot provides a variable amount of dimming voltage to the tube. Most PETs have this turned up to maximum. The capacitor that stores this charge doesn't hold it until the electron gun cools down. The result is a maximum brightness spot with no deflection when PET is turned off. The cure is very simple! An electrolytic capacitor of 20 to 100 ufd at 30 to 50 volts added across the brightness pot (looking from the rear...positive lead to the left side and negative to the right side of the three terminals in a row). This will completely eliminate the danger of burning a hole in the screen.....

P1 Modification

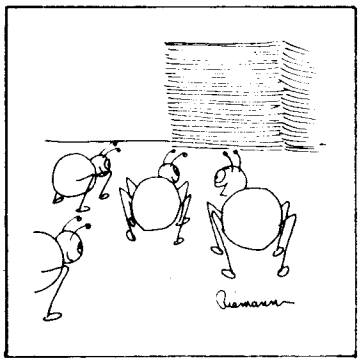
PET owners presently using the Centronics Microprinter P1 will be interested in the following hardware modification sent in by Rick Hoffman of Centronics in Mississauga. It concerns the extra line feeds that occur when LISTing programs on the P1.

To suppress extra line feeds on the P1:

A. Locate IC 1-D
B. Locate IC 2-B
C. Cut PCB trace from 1-D pin 5 to 2-B pin 1



'Give Me All Your $10s, $20s and $50s.'



'Looks Like a Good Program. Climb In, Everybody.'

from Computerworld Magazine